UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERIA

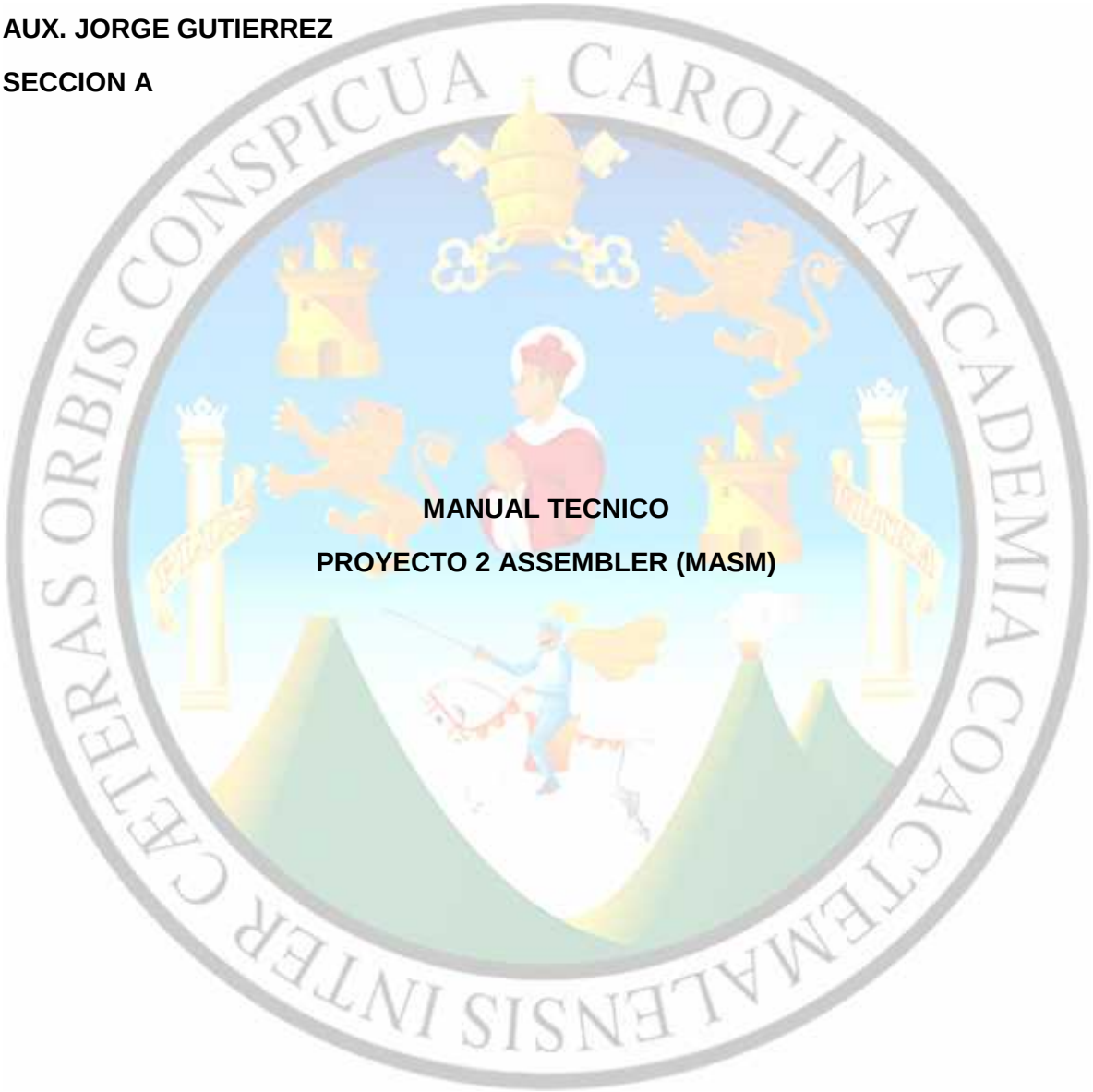
CIENCIAS Y SISTEMAS

ARQUITECTURA DE COMPUTADORES 1

ING. OTTO RENE LEIVA

AUX. JORGE GUTIERREZ

SECCION A



MANUAL TECNICO

PROYECTO 2 ASSEMBLER (MASM)

NOMBRE: OSCAR RENE CUELLAR MANCILLA

CARNET: 201503712

FECHA: 5 DE NOVIEMBRE DE 2017

ENSAMBLADOR USADO EN LA PRACTICA: (MASM)

El Microsoft Macro Assembler (MASM) es un ensamblador para la familia x86 de microprocesadores. Fue producido originalmente por Microsoft para el trabajo de desarrollo en su sistema operativo MS-DOS, y fue durante cierto tiempo el ensamblador más popular disponible para ese sistema operativo. El MASM soportó una amplia variedad de facilidades para macros y programación estructurada, incluyendo construcciones de alto nivel para bucles, llamadas a procedimientos y alternación (por lo tanto, MASM es un ejemplo de un ensamblador de alto nivel). Versiones posteriores agregaron la capacidad de producir programas para los sistemas operativos Windows. MASM es una de las pocas herramientas de desarrollo de Microsoft para las cuales no había versiones separadas de 16 bits y 32 bits.

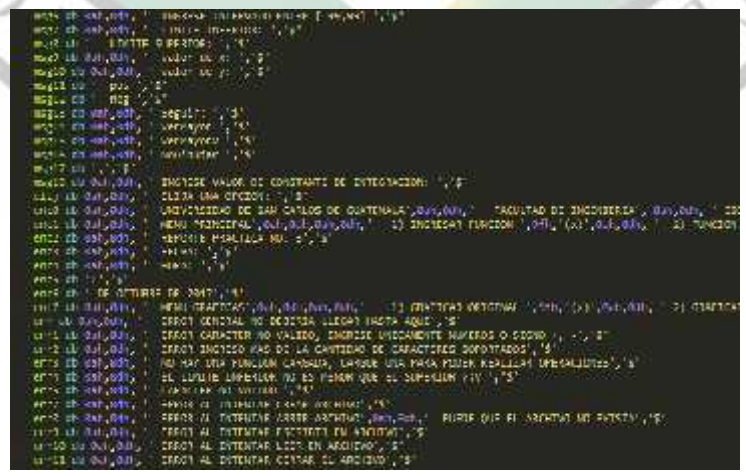
CÓDIGO RELEVANTE DE LA PRACTICA:

Durante la realización de la práctica se utilizaron varios macros, el más utilizado sería el macro llamado “print” con el cual mando como parámetro una cadena a imprimir, se manda al registro AX el @data que representa que se va a escribir una cadena, con el mov ah,09 le indico a mi interrupción 21H que iniciare una impresión de cadena en pantalla y al registro DX le mando la dirección donde se almacena mi cadena a imprimir

```
print macro cadena
  mov ax,@data
  mov ds,ax
  mov ah,09
  mov dx,offset cadena
  int 21h
endm
```

Se declararon todos los mensajes que podían ser mostrados en consola con variables que los representaban de tipo byte:

Cada uno de los cuales es llamado en diferente ocasión y con diferente propósito.



OBTENER UN NUMERO LEIDO DESDE CONSOLA:

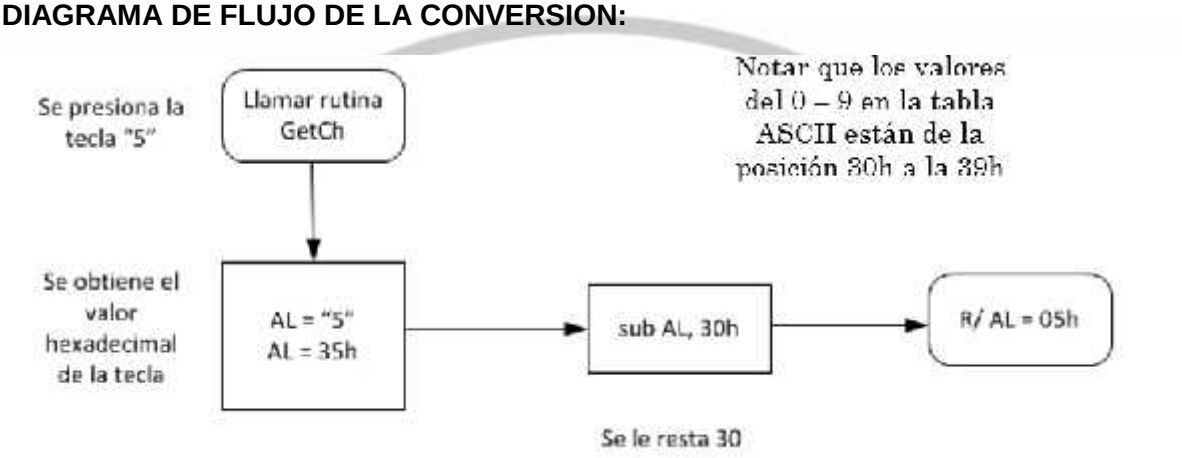
El siguiente algoritmo pide que se ingrese un numero el cual es guardado en un arreglo de tipo byte, el cual posteriormente será mandado al algoritmo de conversión a decimal. Este algoritmo permite el ingreso de signo solo al principio, se podrán ingresar números con el signo + o – solo al comienzo de lo contrario mostrara una advertencia y volverá a solicitar el número, si no se ingresa nada y se ingresa un salto de línea únicamente, este guardara un 0 por default, si se ingresan caracteres incorrectos también se mostrará una advertencia.

```
;-------------------------------------------------
;--------------GUARDAR NUMERO---------------------
;---GUARDA UN NUMERO CON SIGNO, PERMITE EL SIGNO + O - SOLO AL PRINCIPIO
ObtenerNumero macro buffer,lim
LOCAL ON2,Signo2,valido2,FinON2,NoCero
xor cx,cx
xor si,si

ON2:
cmp cx,lim
je Error2 ;error cuando ingresa mas de los caracteres soportados
mov ah,01h
int 21h
cmp al,0dh ;ascii del \n
je FinON2
cmp al,30h ;ascii del 0
jb Signo2
cmp al,39h ;ascii del 9
ja Error1
jmp valido2

Signo2:
cmp cx,0
jne Error1  ;error cuando vuelve a ingresar un + o - luego del inicio
cmp al,2bh ;ascii del +
je valido2
cmp al,2dh ;ascii del -
je valido2
jmp Error1

valido2:
mov buffer[si],al
inc si
inc cx
jmp ON2

FinON2:
cmp cx,0 ;si no ingreso nada, solo el enter, asigno un 0 default
jne NoCero
mov buffer[si],30h
NoCero:
xor si,si
endm
```

CONVERTIR UN NUMERO A DECIMAL:

Al obtener un numero desde consola se pide sea ingresado carácter por carácter, en el siguiente algoritmo se le resta un 30h que es la diferencia entre los números y sus códigos asciis para poder tenerlo en decimal, este número se guarda en una variable de tipo Word. A continuación, se muestra el diagrama de flujo de la conversión y el algoritmo ya implementado en el código:

DIAGRAMA DE FLUJO DE LA CONVERSION:



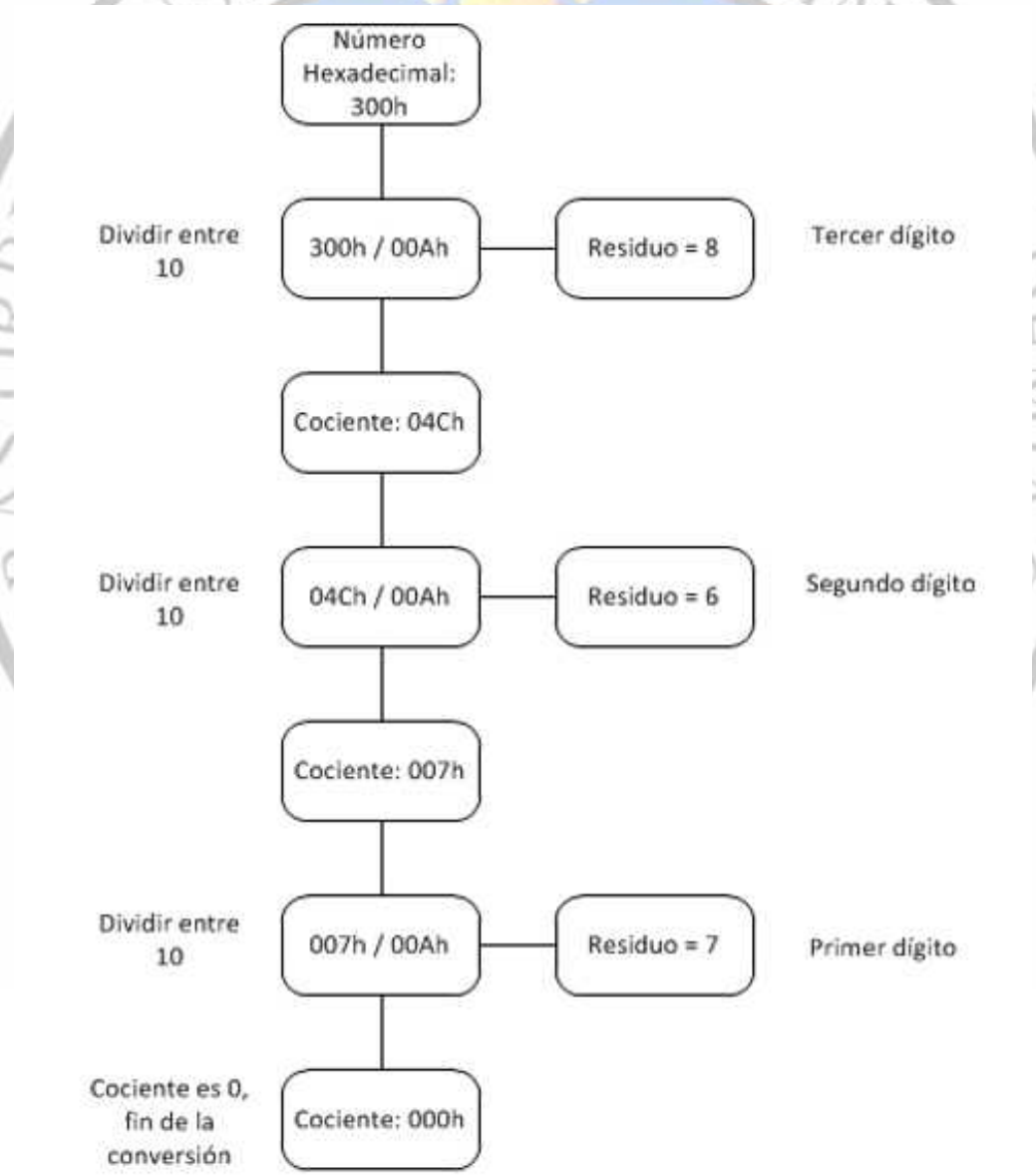
ALGORITMO IMPLEMENTADO EN EL CODIGO:

```
ConvertToDec macro num,buffer
    local FinNumero,SaveNum,OmitirMas,NoComplemento2
    mov si,si
    mov cx,cx
    mov ax,ax
    mov dx,dx
    mov bx,bx
    mov dl,10

    cmp buffer[si],2Dh ;ascii del -
    je OmitirMas
    cmp buffer[si],2Bh ;ascii del +
    jne SaveNum

    mov cx,1 ;bandera para saber si tengo que sacar complemento o no
    OmitirMas:
    inc si ;paso al primer numero

    SaveNum:
    ;print err2
    mov bl,buffer[si] ;muevo el ascii del primer numero al registro BL
    sub bl,30h ;le resto 30h que es la diferencia que hay entre los numeros y sus valores ascii
    inc si ;paso al siguiente digito
    add al,bl   ;sumo lo que llevo en AL contra lo que acabo de encontrar
    cmp buffer[si],20h ;ascii del espacio en blanco
    je FinNumero
    cmp buffer[si],24h ;ascii del $
    je FinNumero
    mul dl  ;multiplico el valor que llevo en AL contra 10
    jmp SaveNum

    FinNumero:
    cmp cx,1
    jne NoComplemento2
    neg ax
    NoComplemento2:
    mov num,ax
    ;print salL
    endm
```

CONVERTIR UN DECIMAL A SUS CODIGOS ASCIIS:

Por el lado contrario, cuando realizamos operaciones con números decimales y necesitamos mostrarlos en pantalla hay que hacer un procedimiento inverso para pasar cada digito del número a su respectivo código ascii y asi poder mostrarlo en pantalla. Para poder mostrar el resultado en pantalla se utiliza el algoritmo de convertir un numero entero a sus respectivos asciis, les sumo 30h que es la diferencia que hay entre los números y sus códigos asciis.

DIAGRAMA DE FLUJO DE LA CONVERSION:



ALGORITMO IMPLEMENTADO:

```
;---------------CONVERTIR RESULTADO--------------------
ConvertirResultado macro res,buffer
xor si,si
xor cx,cx
xor ax,ax
xor dx,dx
mov ax,res
mov dl,0ah
jmp segDiv

segDiv1:
xor ah,ah
segDiv:
div dl
;print msg10
inc cx
push ax
cmp al,00h ;si ya dio 0 en el cociente dejar de dividir
je FinCR
jmp segDiv1

FinCR:
pop ax
add ah,30h
mov buffer[si],ah
inc si
loop FinCR

mov ah,24h ;ascii del $
mov buffer[si],ah
inc si
endm
```

CREACION DEL REPORTE:

Para la creación del reporte se utilizaron los macros para crear archivos y modificar su contenido, para ello se utilizaron los siguientes macros:

```
crear macro ruta,handle
lea dx,ruta
mov ah,3ch
mov cx,00h
int 21h
mov handle,ax
jc Error7
endm
```

```
escribir macro numbytes,databuffer,handle
mov ah,40h
mov bx,handle
mov cx,numbytes
lea dx,databuffer
int 21h
jc Error9
endm
```

```
cerrar macro handle
mov ah,3eh
mov bx,handle
int 21h
jc Error11
endm
```

LIMPIAR ARREGLO:

Para que no se encuentren errores a la hora de reutilizar un arreglo con la basura digital, se utiliza el siguiente macro que limpia un arreglo de tipo byte con un carácter específico y la cantidad de espacios específicas.

```
LimpiarArr macro buffer,caracter,lim
LOCAL Limpiar, FinLA
pop cx
xor si,si
xor cx,cx

Limpiar:
cmp cx,lim
je FinLA
mov buffer[si],caracter
inc si
inc cx
jmp Limpiar

FinLA:
push cx
endm
```

REQUERIMIENTOS MINIMOS:

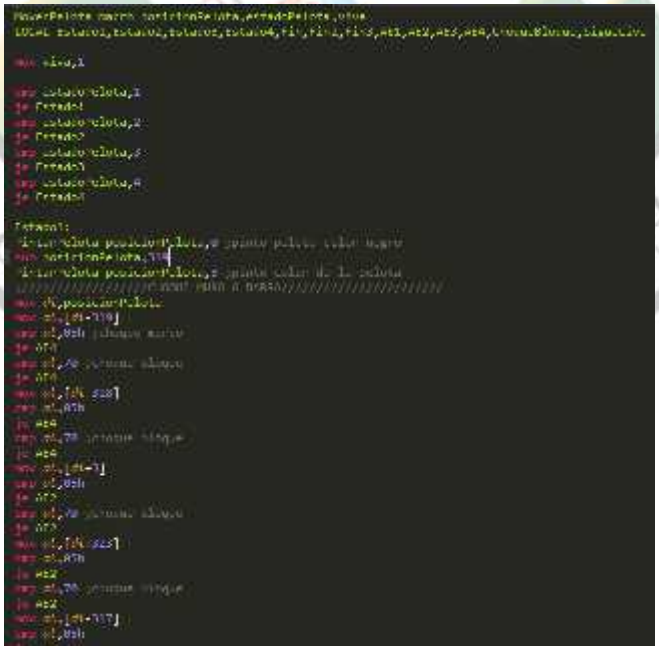
MACROS PARA EL MODO DE VIDEO:

En los siguientes macros se obtiene del modo de video su propiedad y la matriz que proporciona de 320 x 200 para el manejo de las imágenes.

```
;---------REGRESAR AL MODO DE TEXTO-----------------
ModoTexto macro
;Regresar a modo texto
mov ax,0003h
int 10h
endm

;--------MODO GRAFICO--------------------
ModoGrafico macro
;Iniciacion de modo video
mov ax,@data
mov ds,ax
mov ax, 0013h
int 10h
mov ax, 0A000h
mov es, ax
endm
```

MOVIMIENTO DE LA PELOTA:

Para mover la pelota se utilizaron 4 estados para ella, los cuales son movimientos diagonales, para arriba izquierda, arriba derecha, abajo izquierda y abajo derecha. Estos movimientos siempre son de 45 grados por lo cual se aumentaba o disminuía las variables “x” y “y” de la pelota. El algoritmo implementado se muestra a continuación.



VALIDAR CHOQUES:

Para validar si hubo choque con el marco o algún cubo se utiliza el siguiente macro:

```
validarChoque macro noFilas,cuadrosPorFila,ancho,separacion
LOCAL Seguir,Seguir2,Pintar,Fin,Fin2,Comp,Otro,Otro2,Choco
xor cx,cx
xor si,si
xor bx,bx
mov ax,noFilas

mov di,01f7ch ;(320*25)+60 ;inicio de los cubos :D :3

Comp:
push di
mov dl,[di]
cmp dl,0 ;no hay cubo
je Otro2
mov dl,[di]
cmp dl,70 ;hay cubo
jne Fin

mov cx,ancho ;ancho del cubo
Seguir:
mov dl,[di-320] ;arriba
cmp dl,3 ;color de la pelota
je Choco
mov dl,[di-1] ;izquierda
cmp dl,3 ;color de la pelota
je Choco
mov dl,[di+1] ;derecha
cmp dl,3 ;color de la pelota
je Choco

mov dl,[di+319] ;izquierda
cmp dl,3 ;color de la pelota
je Choco
mov dl,[di+321] ;derecha
cmp dl,3 ;color de la pelota
je Choco
```

En el macro anterior se compara cada uno de los pixeles de cada cubo, por ejemplo sus pixeles extremos superiores e inferiores y los laterales, uno por uno y se compara la siguiente posición, se verifica el color que hay en la siguiente posición para saber si la pelota está chocando con alguno de ellos o no, este proceso se repite para cada uno de los cubos hasta que se termina de verificar con todos o cuando se encuentre con cuál de ellos fue que choco, luego de encontrar con cual chocó se procede a eliminarlo y aumentar la variable que lleva control de cuantos cubos fueron eliminados.

PAUSAR JUEGO:

Para pausar el juego se utilizó la interrupción 16h con la cual primero leo del teclado si hay disponible una tecla para leer, luego si la hay, la leo, posteriormente comparo el código de la tecla especial oprimida, si no es otro ESC entonces mantengo leyendo la tecla hasta que se presione otro ESC o la Barra espaciadora para finalizar el juego.

```
mov ah,01h
int 16h ;verificar si hay tecla lista para ser leida
jz jugar
mov ah,00h
int 16h ;leer la tecla

cmp ah,1 ;si es el boton ESC
jne Mover
Mantener:
mov ah,00h
int 16h ;esperar otro ESC o un Espacio
cmp ah,1 ;si es otro ESC
je jugar
cmp ah,57 ;salir del juego
je Fin
cmp al,32h ;a nivel 2
je ANivel2
cmp al,33h ;a nivel 3
je ANivel3
jmp Mantener
```

NIVEL 1:

El nivel 1 del juego está compuesto de 10 bloques y de una pelota, el objetivo de este es destruir los 10 bloques sin que la pelota toque el borde inferior de la pantalla, al lograr romper los 10 bloques nos mandara al nivel 2 y así sucesivamente.

```
Nivel1 macro nivel
LOCAL jugar,Mover,izquierda,derecha,Fin,Mantener,Saltar,SoltarPelota,ANivel2,ANivel3

;Si - Pch  leer hora del sistema(CH-hora; CL-min; DH-seg;
mov ah,2ch
int 21h
mov segundos,dh
mov minutos,cl

mov ch,ch
mov cl,minutos
mov cx,60
mul cx
mov cl,segundos
mov ch,ch
add cx,cx
mov segun,ax

PintarMarco
PintarBarra
;al modificar los parametros de pintar cuoos modificar tambien en verificarChoque
mov perdio,0
mov puntos,0
mov tiempo,0
mov nivel,1
mov lineas,2
mov cuadroslinea,5
mov anchocuadros,30
mov separacioncuadros,10
```

INTERRUPCIONES USADAS EN LA PRACTICA:

Interrupción al sistema 21H:

La mayoría de servicios ó funciones del sistema operativo MS-DOS se obtienen a través de la interrupción software 21H. Es por esto que se le denomina DOS-API: DOS-APPLICATION-PROGRAM-INTERFACE La INT 21H está compuesta por un grupo de funciones. Cuando se accede a la INT 21H, hay que indicar el número de función que queremos ejecutar. La llamada a la INT 21H se realizará como sigue:

- Introducimos en (AH) el número de función a la que deseamos acceder.

- En caso de que deseemos acceder a una sub-función dentro de una función, debemos indicarlo introduciendo en (AL) el número de esa sub-función.

- Llamar a la INT 21H.

Interrupción al sistema 10H (FUNCION 13H):

El modo gráfico 13h nos permite manejar la pantalla como una matriz de 320 píxeles de ancho por 200 píxeles de alto. Cada píxel puede tomar uno de 256 colores, estos colores están definidos en una paleta de colores la cual podemos configurar.

Usamos la interrupción 10h, servicio 0Ch para modificar los píxeles; el parámetro AL más que especificar directamente el color, indica la entrada de la paleta de colores que se debe usar para el píxel ubicado en la fila DX y la columna CX.